Understanding URLs and Request-Response Cycle

1. HTTP & Protocols

- HTTP (HyperText Transfer Protocol): rules for transferring hypertext (HTML, JSON, images, etc.) over the web.
- HTTPS: HTTP over TLS/SSL (encrypted, authenticated).
- Other protocols you might see:
  - FTP (ftp://) for file transfers
  - SMTP (smtp://) for email delivery
  - WebSocket (ws:///wss://) for bi-directional real-time communication

2. URLs & Endpoints

A URL (Uniform Resource Locator) tells you where to find a resource, and how to get it.

- Base URL: The unchanging part of your API's address.
  Example: https://api.example.com/v1
- Endpoint: The path appended to the Base URL to reach a specific resource.
  Example: /users, /products/123
- Full URL: https://api.example.com/v1/products/123?status=open

2.1 URL Components

scheme://host[:port]/path/to/resource?query1=val1&query2=val2#fragment

- Scheme: http, https, ftp, etc.
- Host: domain name or IP (e.g. api.example.com)
- Port: optional (default 80 for HTTP, 443 for HTTPS)
- Path: hierarchical location of the resource
- Query string: key=value pairs, often used for filtering, pagination, sorting
- Fragment: client-side only (e.g. anchors in HTML)

3. HTTP Methods

- **GET**
  - **CRUD Role**: Read
  - **Use**: Retrieve data (e.g., fetch a list of users or a specific post)
- **POST**
  - **CRUD Role**: Create
  - **Use**: Create a new resource (e.g., submit a form, add a comment)
- **PUT**
  - **CRUD Role**: Update
  - **Use**: Replace an entire resource (e.g., update full profile details)
- **PATCH**
  - **CRUD Role**: Update

    - **Use**: Modify part of a resource (e.g., update just the email field)
- **DELETE**
    - **CRUD Role**: Delete
    - **Use**: Remove a resource (e.g., delete a post or user)
- **OPTIONS**
    - **CRUD Role**: Not applicable
    - **Use**: Discover supported HTTP methods for a resource (often used in CORS)
- **HEAD**
    - **CRUD Role**: Not applicable
    - **Use**: Similar to GET but only returns headers (no response body)

4. Request & Response Cycle

1. Client creates a Request (URL, method, headers, optional body, timeout).
2. DNS resolution & TCP/TLS handshake (if HTTPS)
3. Server receives request, processes it.
4. Server sends a Response: status code, headers, body.
5. Client processes the response.

Request Timeout:
- Clients/servers can impose timeouts to avoid hanging.
- On timeout, client usually throws an error and may retry.

5. Components of an HTTP Request

1. Request Line: GET /posts/123?include=comments HTTP/1.1
2. Headers: metadata like Content-Type, Accept, Authorization
3. Body: optional, used in POST/PUT/PATCH, usually JSON

6. Common Best Practices & Extras

- CORS: configure cross-origin access.
- Auth: use OAuth2/JWT in Authorization header.
- Rate Limiting: X-RateLimit-* headers.
- Error Responses: structured JSON with error code/message.
- Logging: tag each request with a unique ID.
- HTTPS and security headers recommended.

Example: Fetching Data (GET)

```
// GET posts from the API
 fetch("https://apis.scrimba.com/jsonplaceholder/posts")
   .then(res => res.json()) // convert response to JSON
   .then(data => console.log(data)); // log the data
```

Example: Sending Data (POST)

```
// POST a new todo to the API
 fetch("https://apis.scrimba.com/jsonplaceholder/todos", {
   method: "POST", // HTTP method
   headers: {
     "Content-Type": "application/json" // specify JSON body
   },
   body: JSON.stringify({
     title: "Buy Milk", // data being sent
     completed: false
   }),
 })
   .then(res => res.json()) // parse the response
   .then(data => console.log(data)); // display result
```